最大公約数の計算アルゴリズム

与えられた自然数の集合に対して、共通する約数（公約数）のうちで最大の自然数を最大公約数と呼びます。ここでは簡単のために二つの自然数 (a, b) を考えて、その最大公約数 $\gcd(a, b)$ の計算方法を考えます。計算のアルゴリズムに入る前に最大公約数の計算例を挙げておきます。

- 例 1
  $(52, 32)$ の最大公約数 $\gcd(52, 32)$ は $52 = 2^2 \times 13$ と $32 = 2^5$ なので $\gcd(52, 13) = 4$
- 例 2
  $(68, 51)$ の最大公約数 $\gcd(68, 51)$ は $68 = 2^2 \times 17$ と $32 = 3 \times 17$ なので $\gcd(68, 51) = 17$

最大公約数の計算は以上の例のように小さな数字の組みに対しては比較的簡単にできます。しかし、ある程度数字が大きくなると機械に頼った計算が必要になってきます。最大公約数の効率的な計算方法としてユークリッド互除法が古くから知られています。このコードではユークリッド互除法を使わない愚直な方法とユークリッド互除法を使った方法の二つを実装しています。

- 愚直な方法
  最初の方法は、最小公約数は少なくとも与えられた自然数の組み (a, b) のうち小さい方の数字 $b(< a)$ 以下である点に注目して、その数字 b から初めて $b, b-1, b-2, \cdots$ と順番に a と b が同時に割り切れるかを確認していきます。以下が疑似コードとなります。

Algorithm 1 simple_gcd

Require: a, b are positive integer. $(a \geq b)$

1: $x = b$
2: **if** $a\%x == 0$ and $b\%x == 0$ **then**
3: return x
4: **end if**
5: $x \leftarrow x - 1$
6: goto 2

- ユークリッド互除法
  二つ目の方法はユークリッド互除法です。具体的なアルゴリズムの前に以下の点に注目します。まず与えられた自然数の組み (a, b) に対する公約数を u とします。するとある自然数 α と β を用いて

$$a = \alpha u \tag{1}$$
$$b = \beta u \tag{2}$$

  と書くことができます。今 $a \geq b$ としておいて a を b で割った商を s、余りを r とすると

$$r = a - bs \tag{3}$$
$$= u(\alpha - \beta s) \tag{4}$$

なので r も u を公約数に持つことがわかります。また逆に b,r の公約数を u' とするとき

$$b = \beta' u' \tag{5}$$
$$r = \gamma' u' \tag{6}$$

と書くことができるが

$$a = (\gamma' + \beta' s)u' \tag{7}$$

なので u' は a の公約数でもあります。

この性質を利用して最大公約数を計算するアルゴリズムが以下のユークリッド互除法です。今、改めて自然の組み (a, b) が与えられているとします。$a \geq b$ としても一般性は失われません。a を b で割った余りを r_0 とすると r_0 は (a, b) と同じ公約数を持ちかつ r_0 は (a, b) どちらよりも小さいです。次に b を r_0 で割った余りを r_1 とします。すると、r_1 は b と r_0 の公約数と同じ約数を持ちかつ r_1 は b と r_0 のどちらよりも小さいです。この作業をくり返いしていくと、以下のような式が得られます。

$$a = bq_0 + r_1 \tag{8}$$
$$b = r_1 q_1 + r_2 \tag{9}$$
$$r_1 = r_2 q_2 + r_3 \tag{10}$$
$$\cdots \tag{11}$$
$$r_n = r_{n+1} q_{n+1} + r_{n+2} \tag{12}$$
$$\cdots \tag{13}$$

この手続きを繰り返していくことで同じ公約数を持つ数字の列 $r_1 > r_2 > r_3 \cdots$ が得られ、最終的に

$$r_{m-1} = r_m q_m \tag{14}$$

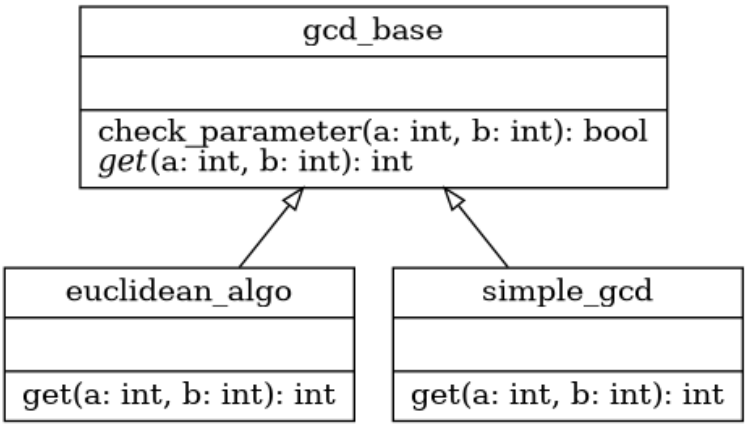
まで行き着きます。この r_m が求める最大公約数となります。このユークリッド互除法を疑似コードで書くと以下のようになります。

Algorithm 2 euclidean_algo

Require: a, b are positive integer. $(a \geq b)$

1: $r = a\%b$
2: **if** $r == 0$ **then**
3: return b
4: **end if**
5: $a, b = b, r$
6: goto 1

以上の二つの方法を用いて最大公約数を計算するコードを Python を用いて実装しました。どちらのコードも最大公約数を計算する機能としては同じなので gcd_base という抽象クラスを定義してそれを継承する simple_gcd と euclidean_algo というクラスを作りました。抽象クラスの方では最大公約数を計算する抽象メソッド get(a,b) を宣言しています。



Listing 1: gcd_base

```
class gcd_base(metaclass = ABCMeta):
    def __init__(self)->None:
        pass
    @abstractmethod
    def get(self,a:int,b:int)->int:
        pass

    def check_parameter(self,a:int,b:int)->bool:
        if a<=0 or b<=0:
            print("a and b must be positive!!")
            return False
        return True
```

この get メソッドは子クラスの方でその実際のアルゴリズムを定義しています。また check_parameter というメソッドも定義しました。こちらは二つの数字が正の整数かどうかを判断するメソッドです。このメソッドは子クラスのどのクラスでも共通なので基底クラスの方に定義しました。

それではこの gcd_base クラスを継承したクラスを見ていきます。

- simple_gcd 最初のクラスは simple_gcd というクラスで実装しました。

Listing 2: gcd_simple

```
class simple_gcd(gcd_base):
    def __init__(self)->None:
        super().__init__()

    def get(self,a:int,b:int)->int:
        if not super().check_parameter(a,b):
            return 0
        if a < b:
            a,b = b,a
        x=b
        while True:
            if a%x == 0 and b%x==0:
                return x
            x-=1
```

こちらのクラスでは与えられた自然数 a,b に対する最大公約数は a,b より小さいことがわかっているので、min(a,b) から順番にマイナス 1 していき最大公約数を計算しています。

- gcd_eucidean

二つ目の方法は gcd_eucidean というクラスで実装しています。

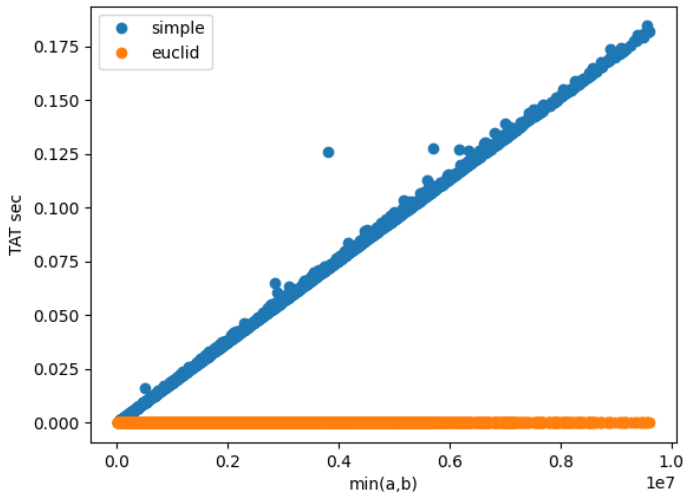
Listing 3: gcd_eucidean

```
class euclidean_algo(gcd_base):
    def __init__(self)->None:
        super().__init__()

    def get(self,a:int,b:int)->int:
        if not super().check_parameter(a,b):
            return 0
        if a < b:
            a,b = b,a
        while True:
            r = a%b
            if r==0:
                return b
            a,b = b,r
```

こちらの方法ではユークリッド互除法を用いて計算しています。

最後に k の二つのアルゴリズムの TAT を計測しました。



参考文献

[1] 辻真吾、下平英寿、「Python で学ぶアルゴリズムとデータ構造」